Ficha 2

Programação Imperativa

1 Arrays

1. Diga, justificando, qual o output de cada um dos seguintes excertos de código C.

(a)

```
int x [15] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
int *y, *z;
y = x; z = x+3;
for (i=0; (i<5); i++) {
  printf ("%d %d %d\n", x[i], *y, *z);
  y = y+1; z = z+2;
}
```

(b)

```
void r (int a[], int n) {
  int i, tmp;
  tmp = a[0];
  for (i=1; (i<n) i++)
    a[i-1] = a[i];
  a[n-1] = tmp;
}
int main () {
  int v [10] = {1,2,3,4,5,6,7,8,9,10};
  int i;
  for (i=0; (i<5); i++) r (v,10);
  for (i=0; (i<5); i++) printf ("%d ", v[i]);
  return 0;
}
```

(c)

```
int s (int a[], int n) {
  int i=0;
  i = i + a[n];
  return i;
}
int main () {
  int v [10] = {1,2,3,4,5,6,7,8,9,10};
```

```
  int i;
  for (i=0; (i<5); i++) r = s (v,i);
  printf ("%d ", r);
  return 0;
}
```

2. Qual o resultado de, na última alínea do exercício anterior, substituir a declaração `int i=0;` por `static int i=0;`

3. Defina uma função `int soma (int v[], int N)` que calcula a soma dos elementos de um array.

4. Defina uma função `void quadrados (int q[], int N)` que preenche o array com os quadrados dos primeiros `N` números naturais.

5. Defina uma função `int somaASCII (char s[])` que calcula a soma dos códigos ASCII de todos os caracteres de uma string.

6. Defina uma função `int contaV (char s[])` que calcula quantas vogais tem uma string.

7. Nas questões que se seguem, assuma que as dimensões das matrizes em causa são constantes (definidas à custa de uma declaração do tipo `#define`).

   (a) Defina uma função `void multM (int r [N][M], int a [N][K], int b[K][M])` que coloca na matriz `r` o produto das matrizes `a` e `b`.

   (b) Defina uma função `void pow (int r[N][N], int a[N][N], int e)` que coloca em `r` o resultado de multiplicar `a` por ela própria `e` vezes ($r = a^e$).

2 Ordenação de vectores

1. Defina uma função `void insere (int v[], int N, int x)` que insere um elemento (`x`) num vector ordenado. Assuma que as `N` primeiras posições do vector estão ordenadas e que por isso, após a inserção o vector terá as primeiras `N+1` posições ordenadas.

2. Usando a função anterior, podemos definir uma função de ordenação de um vector:

```
void iSort (int v[], int N) {
  int i;
  for (i=1; (i<N); i++)
    insere (v, i, v[i]);
}
```

   Apresente uma definição alternativa deste algoritmo sem usar a função `insert`.

3. Defina uma função `int maxInd (int v[], int N)` que, dado um array com `N` inteiros, calcula o índice onde está o maior desses inteiros.

4. Use a função anterior na definição de uma função de ordenação de arrays de inteiros, que vai repetidamente calculando os maiores elementos e trocando-os com o elemento que está na última posição.

5. Apresente uma definição alternativa do algoritmo da alínea anterior sem usar a função `maxInd`.

6. Considere a seguinte definição:

```
void bubble (int v[], int N) {
  int i;
  for (i=1;(i<n); i++)
    if (v[i-1] > v[i]) swap (v,i-1, i);
}
```

   Ilustre a execução da função com um pequeno exemplo. E verifique que após terminar, o maior elemento do vector se encontra na última posição.

7. Use a função `bubble` na definição de uma função de ordenação de vectores que, repetidamente vai colocando o maior elemento do vector no fim.

8. Apresente uma definição alternativa da função pedida na alínea anterior, sem usar a função `bubble`.

9. Uma optimização frequente do algoritmo referido na última alínea, consiste em detectar se o array já está ordenado. Para isso basta que uma das passagens pelo array não efectue nenhuma troca. Nesse caso podemos concluír que o array já está ordenado.

   Incorpore essa optimização nas funções das alíneas anteriores.

10. Defina uma função `void merge (int r [], int a[], int b[], int na, int nb)` que, dados vectores ordenados `a` (com `na` elementos) e `b` (com `nb` elementos), preenche o array `r` (com `na+nb` elementos) com os elementos de `a` e `b` ordenados.

    Usando essa função podemos construir definir o seguinte algoritmo de ordenação:

```
void mergesort (int v[], int n, int aux[]) {
  int i, m;

  if (n>1) {
    m = n/2;
    mergesort (v, m, aux);
    mergesort (v+m, n-m, aux);
    merge (aux, v, v+m, m, n-m);
    for (i=0; (i<n); i++)
      v[i] = aux[i];
  }
}
```

    Note que, se `v` é um vector, `v+m` é o vector (sufixo de `v`) que começa na posição `m`, i.e.,

$$\texttt{v + m} = \texttt{\&(v[m])}$$

11. O algoritmo de quick-sort para ordenação de um vector pode ser descrito da sequinte forma:

- Começa-se por escolher um elemento do array (chamado pivot)
- De seguida, e por trocas entre elementos do vector, obtem-se uma posição `p` tal que o prefixo `0..p-1` só contem elementos menores do que o pivot e o sufixo `p+1..n-1` só contem elementos maiores ou iguais ao pivot (que se encontra na posição `p`)
- Aplicam-se os dois passos anteriores a ambas as partes do vector identificadas acima.

Assumindo então que existe uma função `int partition (int v[], int a, int b)` que realiza a partição do vector referida atrás, podemos codificar este algoritmo da seguinte forma:

```
void qsort (int v[], int n){
  qsortAux (v,0,n-1);
}

void qsortAux (int v[], int a, int b) {
  int p;
  if (a<b) {
    p = partition (v, a, b);
    qsortAux (v, a, p-1);
    qsortAux (v, p+1, b);
  }
}
```

Apresente uma possível definição da função `partition`.